How to Relate Quality and Reuse in Evolving Systems

Dr. Ronald J. Leach
Howard University

This article suggests a model to predict the quality of software developed over time, where the reusable components are also evolving over time.

At NASA Goddard Space Flight Center, we analyzed several safety-critical software systems with sizes ranging from approximately 50,000 to more than 500,000 lines of code (LOC). These systems' architecture included a reusable software core linked to mission-specific software that resulted in complete, unique ground support systems for spacecraft control. Each spacecraft required both new, mission-specific source code and an interface to the reusable core. Because of technology changes, the reusable core itself evolved, increasing in size more than five-fold. The software systems have been somewhat superseded by commercial off-the-shelf products.

The evolving, reusable core had a much lower defect ratio (defects per thousand LOC [KLOC]) for the reusable core than similar systems in the same application domain. The defect ratios of 0.034 and 0.075 for the latest two versions of the reusable core discussed here are far lower than the range of 0.12 to 1.89 for similar systems. The extremely low defect ratio of the core was even more impressive in view of Les Hatton's statement that very few systems, even safety-critical ones, have ever stayed below one defect per KLOC [1].

The Model

What caused the unusually high quality of the evolving, reusable software core? Did the reused part of the core have a low defect ratio simply because it had been operational for so long in a safety-critical domain? If so, we should expect defects per KLOC to follow an exponential distribution of the form a constant times e^{-K*T}.

On the other hand, was the new code good because it underwent stringent analysis and testing? If so, we should expect a linear relationship, reflecting the increase in code size.

These two questions led to a simple model for the number of defects in each release as the sum of an exponential distribution representing defects caused by reused code and a linear expression representing the defects in the new source code. The model is:

$$\text{DEFECTS} = (\text{REUSED} * e^{-K*T} + \text{NEW}) * M * \text{KLOC}$$

where,

T is the time between releases. REUSED and NEW are the percentage of reused and new lines of code in a release, respectively.

KLOC is the size of the release.

The constants *K* and *M* are discussed later.

Calibration of the Model

Calibration of the model requires calculation of the constants *REUSED*, *NEW*, *K*, and *M*. We already know KLOC and want to estimate *DEFECTS*. The overall reuse percentages between releases clustered around 80; hence, the value of *NEW* was .20 and, thus, our the model became:

$$\text{DEFECTS} = (.80 * e^{-K*T} + .20) * M * \text{KLOC}$$

To compute the constant *K*, we entered the defects and the KLOC for each release into an Excel spreadsheet, one entry per release, and then computed the exponent *K* in the exponential distribution, using the Excel LOGEST function to compute *K*. This constant was, to two decimal places, 0.79.

The constant *M* represents the change scale between LOC and *DEFECTS*. To compute *M*, we created a third column in the Excel spreadsheet, representing the difference between the observed number of defects for previous releases and what is predicted by the partial formula $(.80 * e^{-.79*T} + .20) * \text{KLOC}$, one entry per release, and then computed *M* using the Excel SLOPE function. This constant *M* was, to two decimal places, 0.65, making the model for this domain:

$$\text{DEFECTS} = (.80 * e^{-.79*T} + .20) * 0.65 * \text{KLOC}$$

Analysis and Future Work

It is possible to fine-tune the model more than what we presented here. Specific values can be used instead of the averages to improve accuracy. Using different multipliers of *REUSED* and *NEW* can improve estimation, also.

Measures based on exponential distributions frequently are used to assess system quality, reliability, and to stop testing when the expected number of errors remaining meets the objective error rate. As far as we know, their use in conjunction with software reuse to predict faults is new.

Note that applying reuse measurements to evolving code is somewhat controversial. Indeed, Jeffrey Poulin [2] and others recommend against attempting software reuse when the underlying code is not stable. The author recommends it in certain circumstances [3], while others are neutral or do not comment on the issue. Readers are encouraged to provide their experiences via e-mail to the author.◆

Acknowledgement

This research was partially supported by National Science Foundation grant number EIA-0324818.

References

1. Hatton, L. "Does OO Sync With What We Think?" IEEE Software 15.3 (May, 1998).
2. Poulin, J.S. Measuring Software Reuse: Principles, Practices, and Economic Models. Reading, MA: Addison-Wesley, 1997.
3. Leach, R.J. Software Reuse: Methods, Costs, and Models. New York: McGraw-Hill, 1996.

About the Author

Ronald J. Leach, Ph.D., is professor and chair of the department of systems and computer science at Howard University where he performs research on software engineering with special interest in reuse, metrics, and fault tolerance. He has a Bachelor of Science, Master of Science, and doctorate degree in mathematics from the University of Maryland, and a Master of Science in computer science from Johns Hopkins University.

Dept. of Systems and Computer Science
Howard University
Washington, DC 20059
Phone: (202) 806-6650
Fax: (202) 806-4531
E-mail: rjl@scs.howard.edu